2015

ECE 254: Lab 4

PRODUCER CONSUMER PROBLEM

RAMIE RAUFDEEN & SADMAN KHAN

Overview

In this lab, the objective was to use the POSIX API's functionalities to solve the classic producer-consumer problem. The code was then used to generate timings for various factors such as the number of queued messages, initialization of the system, and the data transmission as a whole.

There is potential for bias with the data below as the *printf()* function could cause a delay in the program with its IO wait. This could mean that the data might be a bit skewed.

Timing Data

Initialization Time

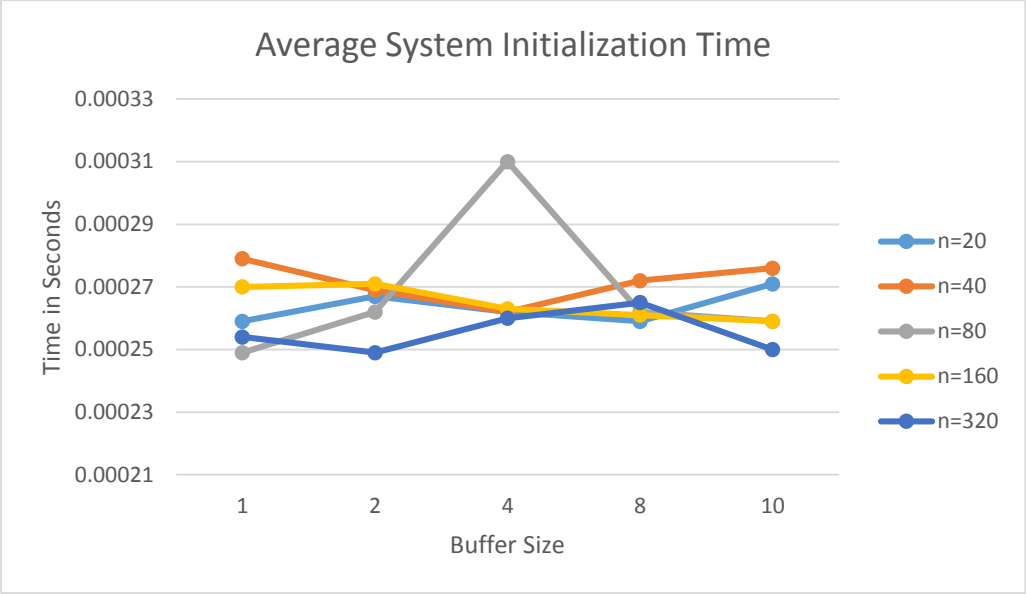
Average System Initialization Time					
N\B	1	2	4	8	10
20	0.000259	0.000267	0.000262	0.000259	0.000271
40	0.000279	0.000269	0.000262	0.000272	0.000276
80	0.000249	0.000262	0.00031	0.000262	0.000259
160	0.00027	0.000271	0.000263	0.000261	0.000259
320	0.000254	0.000249	0.00026	0.000265	0.00025
Standard Deviation of System Initialization Time					
N\B	1	2	4	8	10
20	0.000251	0.000263	0.000244	0.000243	0.000253
40	0.000279	0.000271	0.000256	0.000276	0.000293
80	0.000233	0.000257	0.001186	0.000252	0.000236
160	0.0003	0.000284	0.000254	0.000262	0.000253
320	0.000246	0.000221	0.000258	0.000259	0.000231

Data Transmission Time

Average Data Transmission Time					
N\B	1	2	4	8	10
20	0.001322	0.001347	0.001331	0.001476	0.001564
40	0.001413	0.001446	0.001457	0.001406	0.001399
80	0.001542	0.001525	0.001494	0.0015	0.001491
160	0.001782	0.001718	0.001664	0.001643	0.001627
320	0.002196	0.002108	0.00202	0.002013	0.002094
Standard Deviation of Data Transmission Time					
N\B	1	2	4	8	10
20	0.000425	0.000429	0.00043	0.000493	0.006447
40	0.000492	0.000484	0.000556	0.00045	0.000492
80	0.000469	0.000503	0.00048	0.000477	0.000477
160	0.000595	0.000549	0.00047	0.000514	0.000499
320	0.000681	0.000626	0.000591	0.000594	0.000584

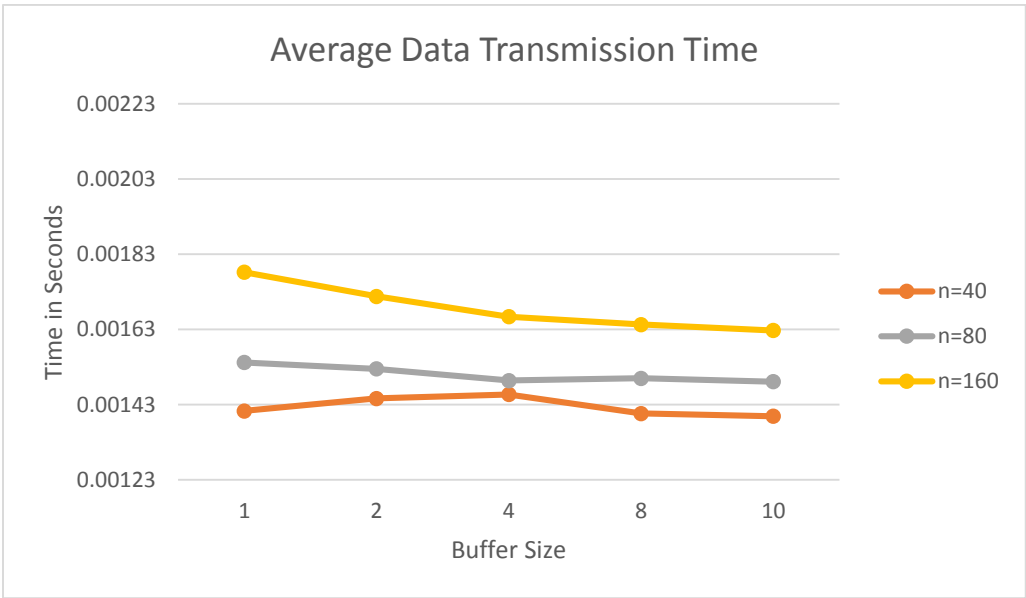
Graphs

System Initialization Time



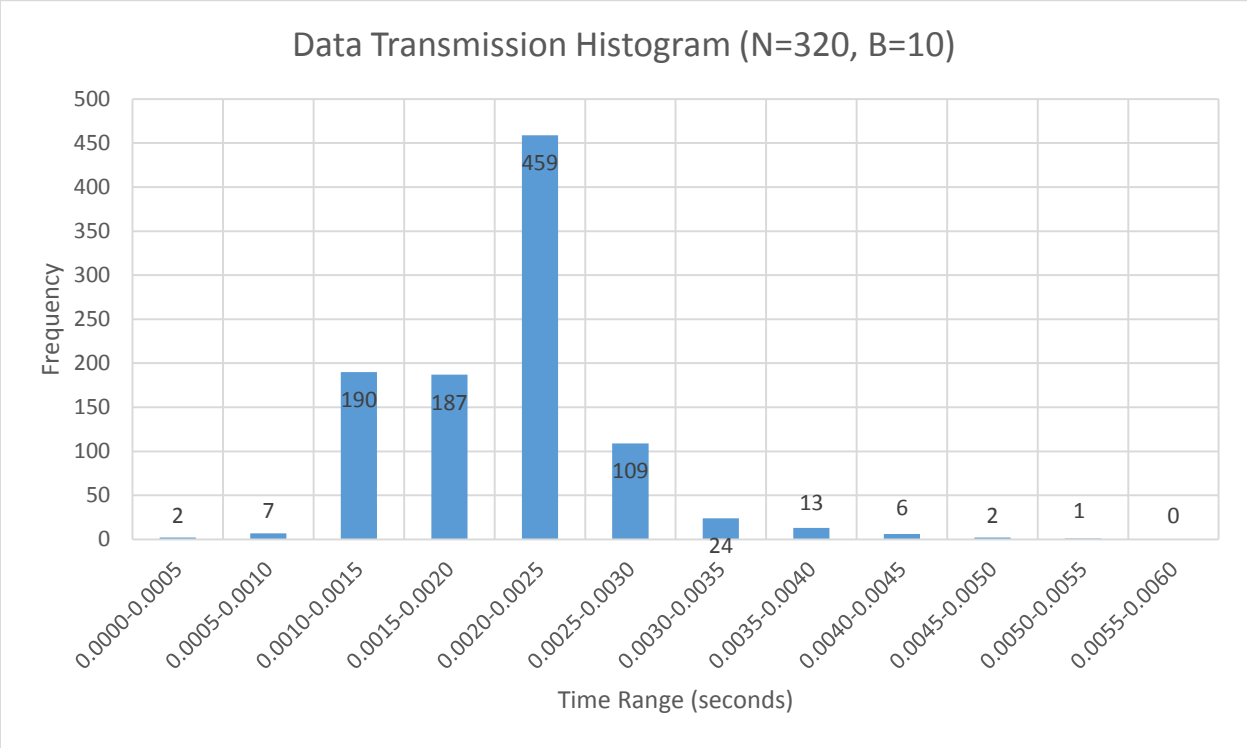
The plot above shows that for any fixed N and a change in buffer size, the resulting initialization time in seconds. Analyzing this graph, one can note there is a steady range of approximately 0.25ms to 0.29ms (excluding the outlier of 0.31ms from n=80) for the initialization time regardless of buffer size. One can conclude from the graph that there is no relationship between the buffer size, number of messages, and the initialization time.

Data Transmission Time



For a fixed N, there is a change in data transmission time as the buffer size changes. In the plot, there is a decrease in average data transmission time as the buffer size increases.

Histogram



Further Discussion & Conclusions

There are three key relationships to conclude from this lab

Altering the number of messages for a fixed buffer size

If you hold B as a constant and increase the N, It will result in longer times for transmission. This makes sense because when the message queue is filled, the producer has to stop producing and wait for the consumer to finish consuming all the elements from the queue. Likewise when the consumer has consumed all the elements from the queue, it has to block it self until there are more messages in the queue again. This causes the system to have to do multiple context switches, which obviously takes a lot of time.

Example

N\B	1
20	0.001322
40	0.001413
80	0.001542
160	0.001782
320	0.002196

Altering the buffer size with a fixed number of messages

If you hold N as a constant and increase the value of B, It will result in shorter times for transmission. This makes sense because more of the messages can be transferred in one go, without the processes having to block themselves. Likewise this results in a lower number of context switches, which makes the transition time shorter over all.

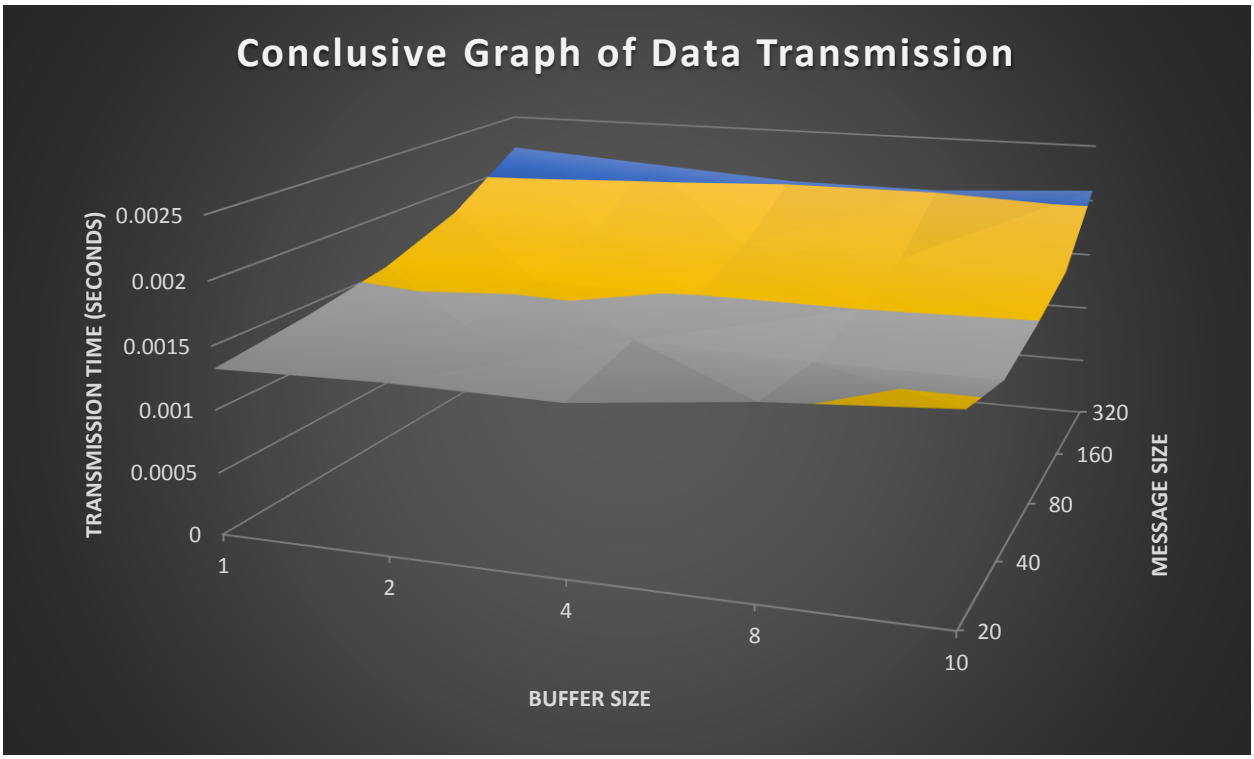
Example

N\B	1	2	4	8	10
320	0.002196	0.002108	0.00202	0.002013	0.002094

Effect on system initialization time

Regardless of what you set as N or B, it will have no effect in system initialization time. This is because the system initiation time is simply the time between right before the fork and the time the producer started producing. In other words, it's a measurement of how long it took to fork a child process.

Conclusion



Using all the data we had for data transmission, we were able to plot a 3-dimensional surface graph to show the relationship between the number of messages sent, buffer size, and the corresponding transmission time. The surface increases as it expands in any direction from the origin of (0,1). Thus, we can conclude that both the number of messages as well as buffer size play a role in the timing of data transmissions for the producer-consumer problem.

Source Code Listing (Appendix)

Producer.c

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  #include <mqueue.h>
4.  #include <sys/stat.h>
5.  #include <sys/types.h>
6.  #include <sys/wait.h>
7.  #include <unistd.h>
8.  #include <errno.h>
9.  #include <sys/time.h>
10. #include <time.h>
11. #include <string.h>
12.
13.
14.
15. int spawn (char* program, char** arg_list, mqd_t qdes) {
16.
17.     arg_list[0] = program;
18.
19.     pid_t child_pid;
20.     child_pid = fork();
21.
22.     if (child_pid != 0) {
23.         return child_pid;
24.     } else {
25.         // ./consume is the compiled file of consumer.c
26.         execvp ("./consume", arg_list);
27.
28.         fprintf(stderr, "an error occurred in exec\n" );
29.         abort ();
30.     }
31. }
32.
33. double time_in_seconds(){
34.     struct timeval tv;
35.     double t1;
36.
37.     gettimeofday(&tv, NULL);
38.     t1 = tv.tv_sec + tv.tv_usec/1000000.0;
39.
40.     return t1;
41. }
42.
43.
44. int main(int argc, char *argv[]) {
45.
46.     /* ensure valid input */
47.
48.     if (argc < 3){
49.         printf("Invalid Inputs\n");
50.         exit(0);
51.     }
52.
53.
54.     /* process the inputs */
55.
56.     int number_of_messages;
```

```
57.     int queue_size;
58.
59.     number_of_messages = atoi(argv[1]);
60.     queue_size = atoi(argv[2]);
61.
62.     if (number_of_messages < 0 || queue_size < 0) {
63.         printf("Invalid inputs");
64.         exit(1);
65.     }
66.
67.     /* setting up and trying to open the message queue for the producer */
68.
69.     struct mq_attr attr;                // message queue attributes
70.     mqd_t qdes;                         // message queue struct
71.     mode_t mode = S_IRUSR | S_IWUSR;    // some kind of mode?
72.
73.     attr.mq_maxmsg  = queue_size;
74.     attr.mq_msgsize = sizeof(int);
75.     attr.mq_flags   = 0;                // a blocking queue
76.
77.     char *qname = "/mailbox_lab4";
78.
79.     //open the queue
80.     qdes  = mq_open(qname, O_RDWR | O_CREAT, mode, &attr);
81.     if (qdes == -1 ) {
82.         perror("mq_open() failed");
83.         exit(1);
84.     }
85.
86.     /* forking a child proccess to be consumer */
87.
88.     double starting_time = time_in_seconds();
89.     pid_t child_pid = spawn("consumer", argv, qdes);
90.     double time_at_first_int = time_in_seconds();
91.
92.     /* sending random numbers from the producer into the message queue */
93.
94.     int i = 0;
95.     int *random_number;
96.     int message_priority = 0;
97.     srand(time(0));
98.
99.     for(i = 0; i < number_of_messages; i++ ){
100.             random_number  = rand() % 100;
101.
102.             //send message to the queue
103.             if(mq_send(qdes, (char*)&random_number, sizeof(int), message_priority) =
    = -1){
104.                 perror("mq_send() failed");
105.             }
106.         }
107.
108.         /* ensure that the consumer has consumed all the ints */
109.
110.         int return_status;
111.         waitpid(child_pid, &return_status, 0);
112.
113.         if(return_status == 0) {
114.             double ending_time = time_in_seconds();
115.             double time_to_initialize = time_at_first_int - starting_time ;
116.             double time_to_consume = ending_time - time_at_first_int;
```

```
117.
118.
119.                printf("Time to initialize system: %f\n", time_to_initialize );
120.                printf("Time to transmit data: %f\n", time_to_consume);
121.
122.            } else {
123.                printf("There has been an error in the child process");
124.                exit(1);
125.            }
126.
127.
128.
129.            /* closing and unlinking the message queue */
130.
131.            if (mq_close(qdes) == -1){
132.                perror("mq_close() failed");
133.                exit(2);
134.            }
135.
136.            if (mq_unlink(qname) != 0) {
137.                perror("mq_unlink() failed");
138.                exit(3);
139.            }
140.
141.            return 0;
142.
143.
144.
145.        }
```

Consumer.c

```c
#include <stdbool.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <mqueue.h>
#include <sys/stat.h>
#include <signal.h>

int main(int argc, char *argv[]) {

    /* processing inputs */
    int number_of_messages = atoi(argv[1]);
    int queue_size = atoi(argv[2]);


    /* opening the message queue for the consumer */

    mqd_t qdes;
    mode_t mode = S_IRUSR | S_IWUSR;
    struct mq_attr attr;

    attr.mq_maxmsg  = queue_size;
    attr.mq_msgsize = sizeof(int);
    attr.mq_flags   = 0;    /* a blocking queue  */

    char *qname = "/mailbox_lab4";

    //open the queue
    qdes  = mq_open(qname, O_RDONLY, mode, &attr);
    if (qdes == -1 ) {
        perror("mq_open()");
        exit(1);
    }


    /* consume elements */

    int i;
    int message_priority = 0;
    for(i = 0; i < number_of_messages; i++){
        int recieved_message;

        //receive message from the queue
        if (mq_receive(qdes, (char *)&recieved_message, sizeof(int), message_priority)
== -1){
            printf("mq_receive() failed\n");
            return 1;
        } else {
            printf("%d is consumed\n", recieved_message);
        }

    }

    /* closing the message queue on the consumer side */

    if (mq_close(qdes) == -1) {
        perror("mq_close() failed");
```

```
58.         exit(2);
59.     }
60.
61.     return 0;
62.
63. }
```